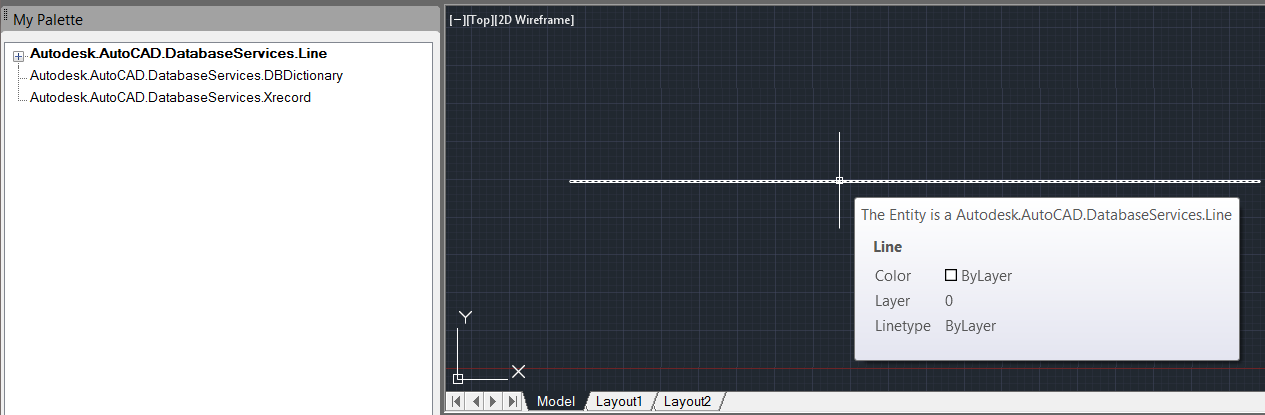
# Lab 6 – PointMonitor

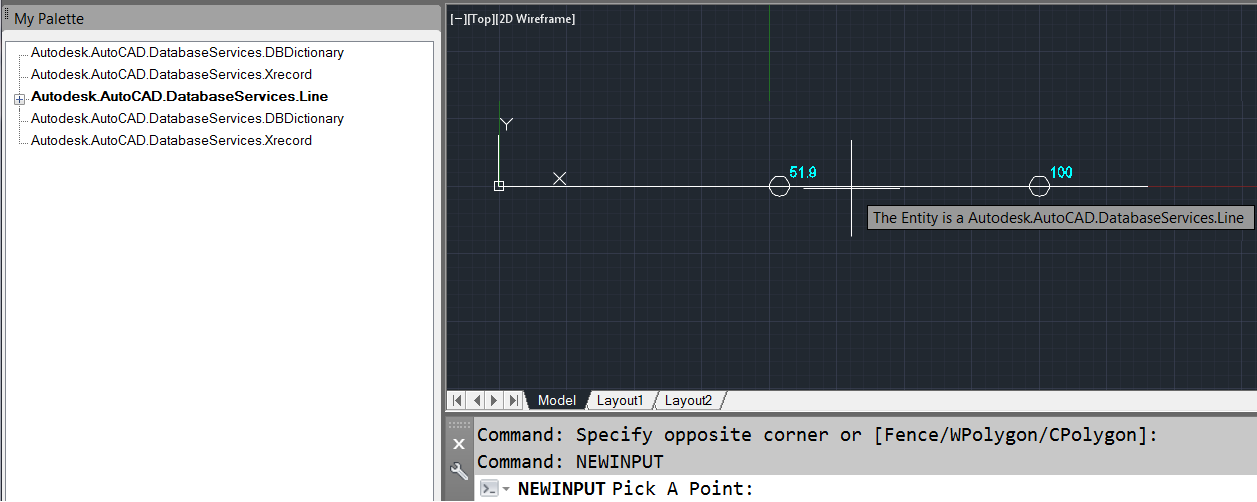
The steps for this lab are in two commands and two functions. One of the commands and one of the functions are created using the steps in the lab. The other command and function already exist and are not part of the lab steps. (These steps have been done in previous labs).

The new command (steps 1-3) will have a name like “addpointMonitor”. The name of the new function will have a name like “MyPointMonitor” (steps 4-26) and will be the call back for the PointMonitor added in the “addpointMonitor” command. In this callback that is called when the cursor moves it will get an object called FullSubentityPath from the PointMonitorEventArgs passed into the function. (If the cursor is over an entity). The FullSubentityPath consists of an array of object IDs and a SubentityId object. The subentity ID object contains the graphics system marker of the subentity and the type of the subentity (that is, edge, face, vertex).

The code will append a ToolTip with the entity type the cursor is over. In the callback you will also highlight the object entry in the TreeView created in previous labs.



The pre existing command is “NewInput” (steps 27-30). In these steps you will use another PointMonitor but it will be added and then removed before the command ends. During the Editor GetPoint function the PointMonitor is used to draw temporary graphics at certain positions along the entity. In the lab’s existing function “MyInputMonitor” steps (31-44) you will use the Curve class GetPointAtDist method with values from the Extension Dictionary to control where the temporary graphics are drawn. To keep the size of the temporary graphics consistent regardless of the zoom value you use GetNumPixelsInUnitSquare. To create the temporary graphics you use methods of the PointMonitorEventArgs passed into the event callback.



The comments are the steps for the lab. You can copy the comments and code below to the cs file in your existing project (need to copy inside of an existing class) or open the Lab6 project that already has these procedures and steps.

// Start of Lab6

// 1. Use the CommandMethod attribute and create a command named "addpointmonitor"

// name the function something like startMonitor

// Note: put the closing curley brace after step 3

// 2. Declare an Editor variable named ed. Instantiate it using the Editor property

// of the Application.DocumentManager.MdiActiveDocument

// 3. Connect to the PointMonitor event. Use the PointMonitor

// event of the editor variable created in step 2. Use

// += and use the new statement and create a new PointMonitorEventHandler. For

// the target parameter use the name of a function we will create in step 4.

// (MyPointMonitor).

// 4. Create a Public function named MyPointMonitor. This is the functions that

// will be called everytime the mouse moves. (The name needs to be

// the name used in the Delegate parameter of step 3). The first parameter is an

// object. (Use sender as the name of the Object). The second parameter is

// a PointMonitorEventArgs. (Use e as the name of the PointMonitorEventArgs)

// Note: Put the closing curley brace after step 26

// 5. Declare an array of the Type FullSubentityPath type. For the name

// of the array use something like fullEntPath. Instantiate it by

// making it equal to the GetPickedEntities method of the Context

// property of the PointMonitorEventArgs passed into the Sub

// 6. Use an "if" statement and test the Length property of the array

// created in step 5. Make sure it is greater than zero.

// Note: Put the closing curley brace after step 26

// 7. Declare a variable named trans as a Transaction. Instantiate it by makine it

// equal to the return of the StartTransaction method of the TransactionManager

// of the current database.

// Application.DocumentManager.MdiActiveDocument.Database.TransactionManager.StartTransaction

// 8. Create a Try Catch block.

// Note: Put the catch and finally statements after step 25

// (enclose step 26 in the finally block).

// Put the closing curley brace for the try after step 26

// 9. Declare a variable as an Entity. Instantiate it using the

// GetObject method of the transaction created in step 7. For

// the ObjectId parameter use the first element in GetObjectIds()[0]

// of the zero element in the array of FullSubentityPath created in

// step 5 open the Entity for read

// 10. Add a tooltip by using the AppendToolTipText method of the

// PointMonitorEventArgs passed into the function.

// Use something like this for the string argument:

// "The Entity is a " + ent.GetType().ToString()

// 11. Use an "if" statement and Check that the palette (myPalette) has

// been created. (== null) f it is null return.

// 12. The following steps will make the text of the entry for a DBEntity

// in the palette created in Lab4 Bold. Declare a variable named fontRegular

// as a System.Drawing.Font. Instantiate it by making it equal to a New

// System.Drawing.Font. For the arguments use the following:

// "Microsoft Sans Serif", 8, System.Drawing.FontStyle.Regular

// 13. Declare a variable named fontBold as a System.Drawing.Font.

// Instantiate it by making it equal to a New

// System.Drawing.Font. For the arguments use the following:

// "Microsoft Sans Serif", 8, System.Drawing.FontStyle.Bold

// 14. Use the SuspendLayout() method of the treeView1 created in Lab4 to

// wait until after the steps below are processed. Use the "this" Keyword to

// get to the Palette (myPalette) with the treeView.

// 15. Here we will search for an object in the treeview control so the font

// can be chaged to bold.

// Create a foreach statement. Use node for the element name and the type is

// System.Windows.Forms.Treenode. The group paramater is the Nodes in the TreeView.

// (myPalette.treeView1.Nodes)

// Note: put the closing curley brace below step 21.

// (In this foreach if the cursor is over an entity and the entity is

// an entry in the TreeView it will be highlighted.

// 16. use an "if else" and see if the Tag property of the node

// is equal to the ObjectId of the entity declared in step 9.

// (Use ToString for the comparison)

// Note: put the "else" after step 19

// Put the closing curley brace after step 21.

// 17. If we get here then the node is the one we want.

// Use "!if" (not) and use the Equals method of the

// System.Drawing.Font variable created in step 13. For

// the Object parameter use the NodeFont property of the node.

// Note: Put the closing curley brace after step 19

// 18. Make the NodeFont property of the node equal to the

// System.Drawing.Font variable created in step 13.

// 19. Make the Text property of the node equal to the

// node.Text property.

// 20. If we get here then the node is not the node we want.

// Use "!if" (not) and use the Equals method of the

// System.Drawing.Font variable created in step 12. For

// the Object parameter use the NodeFont property of the node.

// Note: Put the closing curley brace after step 20.

// 21. Make the NodeFont property of the node equal to the

// System.Drawing.Font variable created in step 12

// 22. Now it's time to recalc the layout of the treeview. Use the

// ResumeLayout() method of the TreeView. Use the this Keyword to

// get to the Palette (myPalette) with the treeView.

// 23. Refresh the TreeView with the Refresh method() of the TreeView.

// 24. Update the TreView with the Update method() of the TreeView.

// 25. All is ok if we get here so Commit the transaction created in

// step 7.

// 26. Whatever happens we must dispose the transaction. (This is

// in the Finally block). Use the Dispose method of the

// transaction created in step 7.

// Note: continue to step 27 in the NewInput function. You could also build and

// test the addPointmonitor command before completing the following steps.

[CommandMethod("newInput")]

public void NewInput()

{

// start our input point Monitor

// get the editor object

Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;

// now add the delegate to the events list

ed.PointMonitor += new PointMonitorEventHandler(MyInputMonitor);

// 27. Need to enable the AutoCAD input event mechanism to do a pick under the prevailing

// pick aperture on all digitizer events, regardless of whether a point is being acquired

// or whether any OSNAP modes are currently active. Use the TurnForcedPickOn method

// of the Editor created above. "ed"

// 28. Here we are going to ask the user to pick a point. Declare a variable as a

// PromptPointOptions. Instantiate it by creating a new PromptPointOptions

// for the Message parameter using something like "Pick A Point : "

// 29. Declare a variable as a PromptPointResult. Instantiate it using the GetPoint

// method of the editor created above. "ed". Pass in the PromptPointOptions created

// in step 28.

// if ok

//if (getPointResult.Status == PromptStatus.OK)

//{

// // ' do something...

//}

// 30. Now remove our point monitor as we are finished With it.

// Use the PointMonitor property of the Editor created above. "ed".

// use -= and use new with a PointMonitorEventHandler for the Object

// parameter use "MyInputMonitor"

// Continue to step 31 in the MyInputMonitor function.

}

public void MyInputMonitor(object sender, PointMonitorEventArgs e)

{

if (e.Context == null)

{

return;

}

// first lets check what is under the Cursor

FullSubentityPath[] fullEntPath = e.Context.GetPickedEntities();

if (fullEntPath.Length > 0)

{

// start a transaction

Transaction trans = Application.DocumentManager.MdiActiveDocument.Database.TransactionManager.StartTransaction();

try

{

// open the Entity for read, it must be derived from Curve

Curve ent = (Curve)trans.GetObject(fullEntPath[0].GetObjectIds()[0], OpenMode.ForRead);

// ok, so if we are over something - then check to see if it has an extension dictionary

if (ent.ExtensionDictionary.IsValid)

{

// open it for read

DBDictionary extensionDict = (DBDictionary)trans.GetObject(ent.ExtensionDictionary, OpenMode.ForRead);

// find the entry

ObjectId entryId = extensionDict.GetAt("MyData");

// if we are here, then all is ok

// extract the xrecord

Xrecord myXrecord;

// read it from the extension dictionary

myXrecord = (Xrecord)trans.GetObject(entryId, OpenMode.ForRead);

// 31. ' We will draw temporary graphics at certain positions along the entity

// Create a "foreach" loop. For the element use a TypedValue

// named myTypeVal. For the group use the Data property of the Xrecord

// instantiated above. "myXrecord".

// Note" put the closing curley brace after step 44.

// 32. Use an "if" statement and see if the TypeCode of the TypedValue

// is a real. (Use DxfCode.Real as the test).

// Note: Put the closing curley brace after step 44.

// 33. To locate the temporary graphics along the Curve

// to show the distances we need to get the point along the curve.

// Note: The value of the TypedValue will be 51.9, 100.0 and 320.6.

// These values were added in Lab 5. Because of this the entity that you

// hover over needs to be at least 51.9 units long. (keep this in mind

// when you test this).

// Declare a vaiable as a Point3d object. Instantiate it using

// the GetPointAtDist method of the ent instantatied above. "ent"

// For the Value parameter using the Value property of the TypedValue

// 34. We need to work out how many pixels are in a unit square

// so we can keep the temporary graphics a set size regardless of

// the zoom scale. Declare a variable as a Point2d name it something

// like "pixels". Instantiate it using the GetNumPixelsInUnitSquare method

// of the current Viewport. (Pass in the Point3d created in step 33).

// Use: e.Context.DrawContext.Viewport.GetNumPixelsInUnitSquare()

// 35. We need some constant distances. Declare a variable as a double

// named something like "xDist". make it equal to 10 divided by the

// X property of the Point2d variable created in step 34.

// 36. Declare a variable as a double named something like "yDist".

// make it equal to 10 divided by the Y property of the Point2d variable

// created in step 34.

// 37. Draw the temporary Graphics. Declare a variable as a Circle

// instantiate is by creating a new Circle. Use the Point3d variable

// created in step 33 for the for the center. For the normal use

// Vector3d.ZAxis. For the radius use the double from step 35.

// 38. Use the Draw method to display the circle. (Pass in the circle).

// Use: e.Context.DrawContext.Geometry.Draw()

// 39. Here we will add more temporary graphics. (text). Declare

// a variable as DBText. Instantiate it by creating new DBText.

// 40. Always a good idea to set the Database defaults With things like

// text, dimensions etc. Use the SetDatabaseDefaults method of the DBText

// from step 39

// 41. Set the position of the text to the same point as the circle,

// but offset by the radius. Use the Position property and make it

// equal to the Point3d created in step 33 plus a New Vector3d. For

// the X parameter use the Double from step 35. For the Y parameter use

// the double from step 36. For Z just use zero.

// 42. Use the data from the Xrecord for the text. Use the TextString

// property of the DBText created in step 39. Make it equal to the

// Value of the TypedValue. (use ToString)

// 43. Make the Height of the DBText equal to the double created in

// step 36

// 44. Use the Draw method to display the text. (Pass in the DBText).

// Use: e.Context.DrawContext.Geometry.Draw()

// Note: The backgound color may impact the display of the temporary

// text. (it displays as white in this example, so may need to change

// the background color to see it, or change the color of the DBText)

// End of Lab6

}

// all ok, commit it

trans.Commit();

}

catch (Exception ex)

{

}

finally

{

// whatever happens we must dispose the transaction

trans.Dispose();

}

}

}